it is embodied by the contents and configuration of the storage device **120**. A persisted offline (i.e., non-executing) copy of the system may be called, herein, a "system image."

[**0037**] **FIG. 1** shows, for example, three artifacts (**130**, **140**, and **150**) stored on the storage device **120**. Herein, "software artifacts" or simply "artifacts" are collections of individual software items stored on the computer-storage device. **120**. Portions of these items may be stored in various system stores including file systems, databases, configuration registries, etc. Those artifacts represent the system-embodying content and configuration. A computer's storage device may have a multitude of artifacts. A system image of a computer contains a multitude of artifacts.

[**0038**] Unlike a conventional software-based computer, the artifacts of the computer **102** are not merely an accumulation of bits resulting from series of ad hoc events during the lifetime of the computer. Rather, each of the artifacts of the computer **102** are associated with at least one manifest. For example, systems artifact **130** has its associated manifest **132** stored therewith the artifact or at some derivable or known-location on the storage device **120**. Artifacts **140** and **150** have their associated manifests, **142** and **152** respectively.

[**0039**] These artifacts are called "self-describing artifacts" because each of the artifacts (via its associated manifest of metadata) describes itself. Rather than being procedural (e.g., a list of actions to be performed), the self-describing metadata descriptions are a declarative description of the desired state of the artifact.

[**0040**] Each description is a complete prescriptive record of the state necessary for the artifact to be consistent and correct. By analogy while a set of directions is procedural, a precise address is declarative and more powerful in the sense that it allows new computation; for example, one can use the address to determine a new set of directions for a different starting point. The declarative description includes a record of all of the artifact's interdependencies and interrelationships with other components of the system.

[**0041**] These metadata descriptions effectively bridge low-level and high-level software abstractions. Low-level software abstractions include, for example, particular artifacts (e.g., load modules) on a storage device and particular processes executing on the computer. High-level software abstractions include, for example, running applications programs and families of applications. High-level software abstractions may also include the running operating system (such as OS **112**) and its elements.

[**0042**] As depicted in **FIG. 1**, the computer **102** has, in the memory **110**, three oversight and managerial functional components that utilize the self-components include a self-describing artifact manager **160**, an execution gatekeeper **162**, and a systems verifier **164**.

[**0043**] While each of these functional components are shown separately in **FIG. 1**, their functionality may be combined into fewer components or expanded to additional components. Furthermore, these functional components may be part of the computer's OS **112** or they may be part of a non-OS component of the computer **102**.

[**0044**] The self-describing artifact manager **160** manages the self-describing artifacts on the storage device **120**. As part of that management, the manager may facilitate persistence and structuring of artifacts. The manager may ensure the maintenance of the association between each artifact and its manifest. The manager may ensure the consistency of an artifact to the description in its manifest. Furthermore, the manager may update self-describing artifacts in response to changes in configuration of the system.

[**0045**] The self-describing artifact manager **160** may assist in the optimization on the loading of artifacts from the storage device **120**. The self-describing artifact manager may take a larger view of the overall operation of the applications and the OS of the computer. Based on this view, the self-describing artifact manager **160** may determine which artifacts (e.g., load modules) will be combined in processes for a particular application. The manager may then combine artifacts into a smaller, and presumably optimized, number of artifacts. Similarly, using the system manifest (which contains declarative descriptions of the entire system) to determine which applications will be invoked soon, the self-describing artifact manager can encourage the start of some applications before they are actually invoked.

[**0046**] The execution gateway **162** clears an application (and possibly other executable components) for invocation. When invocation of an application is pending, the execution gatekeeper **162** examines its associated self-describing artifacts. The declarative descriptions in the self-describing artifacts may specify explicit static or dynamic conditions that are required for the associated application. An example of typical explicit conditions is a list of necessary components, which must exist on the system for the successful execution of the application. Another example of typical explicit conditions is a list of applications and system components that must have been launched (or are in a specified current state) before an application itself is allowed to launch.

[**0047**] The gatekeeper examines the current conditions and if they meet the requirements specified in the declarative descriptions of the associated artifacts, then the gatekeeper allows invocation of the application. Otherwise, then the gatekeeper does not allow invocation of the application.

[**0048**] With the governance of the execution gatekeeper **162**, no code will execute on the computer **102** unless the code is described in an associated manifest. In one embodiment, only code described in associated manifests and signed by trusted software publishers may be installed or run on the computer **102**.

[**0049**] In addition, the execution gatekeeper **162** may perform audits on the integrity of the system as a check against external modification (e.g., by way of innocent data corruption or malicious attacks). The audit is based upon the manifests of the self-describing artifacts of the system.

[**0050**] The system verifier **164** performs one or more verifications on the self-describing artifacts. It may perform this in response to a manual request to do so, in response to an action (e.g., installation of new software), in reponse to new information becoming available, and/or as scheduled to do so. Furthermore, verifications made by the system verifier **164** are based, at least in part, upon information gathered from an examination of the manifests of the self-describing artifacts.